HOCHSCHULE HEILBRONN

TECHNIK WIRTSCHAFT INFORMATIK

Ruprecht-Karls-Universität Heidelberg, Hochschule Heilbronn

Studiengang Medizinische Informatik

Bachelorarbeit

„Implementierung des Bresenhams-Algorithmus“ für die Benutzung in einem Rekonstruktionsalgorithmus

---

Wessam Kais, 01.03.2011

Referent: Prof. Dr. Oliver Kalthoff

Korreferent: Prof. Dr. Rotraut Laun

Kurzfassung

In der Medizin nimmt die medizinische Bildgebung eine wichtige Rolle ein. Die Reaktionszeit chemischer Reaktionen im menschlichen Körper kann sehr kurz sein, so dass man diese Reaktionen nur schwer beobachten kann. Somit kann die Geschwindigkeit der bildgebender Verfahren in der Diagnostik eine ausschlaggebende Funktion haben. Des Weiteren sind bei der Bildgebung eingesetzte Strahlen potentiell gefährlich. Je länger die Aufnahme dauert, desto mehr Schaden kann angerichtet werden.
Im Hinblick auf die Geschwindigkeit der bildgebenden Verfahren haben die unterschiedlichen Bildrekonstruktionsmethoden eine zentrale Bedeutung.

Für die Benutzung in einem tomographischen Rekonstruktionsalgorithmus und somit für die Verbesserung der Geschwindigkeit und die Verringerung der eingesetzte Strahlen der Bildgebung, soll in dieser Arbeit der Bresenham-Algorithmus in Matlab implementiert werden.
Mit Hilfe des Algorithmus werden alle Voxel eines Objektes ermittelt, die innerhalb des bestrahlten Bereichs liegen.
Das Ergebnis des Algorithmus, sprich die Menge und die Koordinaten der bestrahlten Voxel, kann dann als Input für die Berechnung des relativen Volumenanteils oder für die Berechnung der Strahllänge im Voxel verwendet werden.

Inhaltsverzeichnis

1. Einleitung

Die bildgebende Diagnostik ist heute aus der klinischen Praxis nicht mehr wegzudenken. Den Anfang machte *Wilhelm Conrad Röntgen*, der 1895 die unsichtbaren Strahlen entdeckte. Seit den 70er Jahren findet der Begriff „bildgebendes Verfahren“ in der Medizin immer mehr an Bedeutung. [2]

Die Computertomographie (eine leistungsfähige zerstörungsfreie Auswertungstechnik zur Herstellung von 2-D und 3-D-Schnittbilder eines Objektes aus Röntgenbilder) ist eines der wichtigsten Verfahren in der medizinischen Bildgebung. Aus einem CT-Bild wird ermöglicht, die Form, die Dichte und die internen Defekte eines Objektes abzulesen.

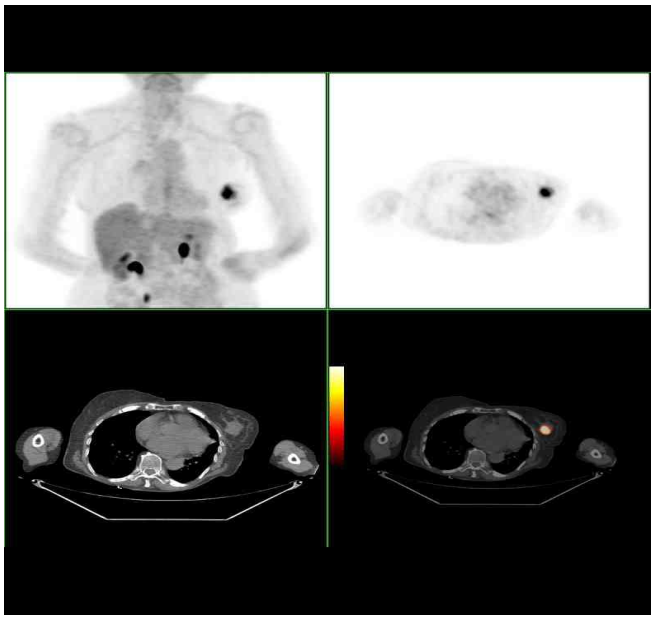


Abbildung 1: Brustkrebserkrankung in der PET/CT

Quelle: http://upload.wikimedia.org/wikipedia/commons/b/b6/Breast_Cancer.jpg

Die Positronen-Emissions-Tomographie (PET) ist ein weiteres bildgebendes Verfahren der Nuklearmedizin. „PET ist bei der richtigen Fragestellung ein hochsensitives Verfahren; Aktivitätsanreicherungen lassen sich jedoch anatomisch nicht immer gut lokalisieren, da in PET Bildern in erster Linie Stoffwechselprozesse aufgezeigt werden; hinzu kommt die begrenzte Ortsauflösung von etwa 4–6 mm. Ein PET/CT kombiniert die hohe Ortsauflösung (von bis zu 0,35 mm) und detailreiche Anatomiedarstellung des CT mit den hochsensitiven

Stoffwechselinformationen aus der PET.“ [*Quelle: Wikipedia, Positronen-Emissions-Tomographie,*[1]*, 20.02.2011*]

Abbildung 2 zeigt, wie das Objekt auf einer fahrbaren Liege zwischen einer Strahlenquelle und einer bildgebendes-Systems platziert ist. Die Liege und das Imaging-System sind an einem Computer angeschlossen, so dass Röntgenbilder gesammelt werden und gleichzeitig mit der Position des Objekts gespeichert werden können. Am Computer wird es ermöglicht aus den Rohdaten Querschnittsbilder zu produzieren und diese zu einem 3D-Volumendatensatz zusammenzufassen. [1]

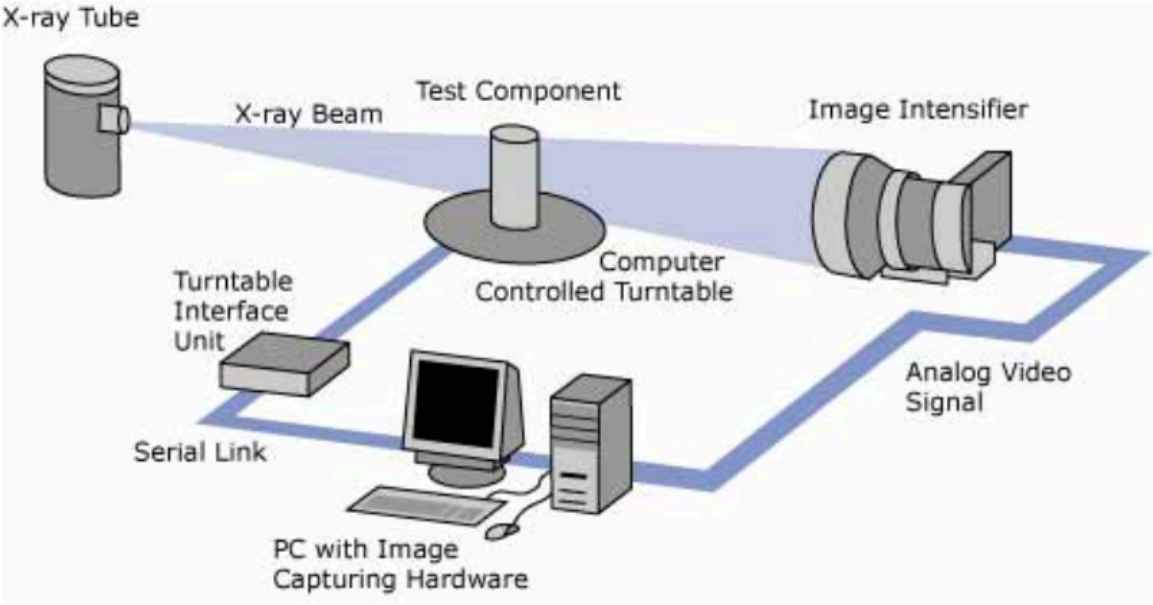


Abbildung 2: Eine schematische Darstellung eines CT-System

Quelle: http://www.ndt-ed.org/...[2]

An dieser Stelle kommen auch die Bildrekonstruktionsalgorithmen ins Spiel. Mit Hilfe der gefilterten Rückprojektion[3] werden die vom PET oder CT aufgenommene Rohdaten in Schnittbilder umgerechnet. Deshalb nennt man das Verfahren auch Schnittbildverfahren.

1 http://de.wikipedia.org/wiki/Positronen-Emissions-Tomographie - Vom_PET_zum_PET.2FCT

2 http://www.ndt-ed.org/EducationResources/CommunityCollege/Radiography/AdvancedTechniques/computedtomography.htm

3: Die gefilterten Rückprojektion ist eine Bildrekonstruktionsmethode.

1.1. Motivation

Die Motivation für diese Arbeit besteht darin, den Lesern und Leserinnen die Wichtigkeit eines Algorithmus im Bezug auf die Bildrekonstruktionsverfahren und somit auf die Bildgebung in der Medizin aufzuzeigen.
Die Strahlendosis ist eine der größten Nachteile der Computertomographie. Man kann allerdings die Verwendung mit repräsentativen und gehaltvollen PET/CT-Bilder verteidigen, obwohl die Computertomographie mehr Strahlung als andere bildgebende diagnostische Verfahren liefert. Bei einer Röntgenaufnahme trifft den Patienten bis zu 1000-fach weniger Strahlenmenge als bei einer CT.[4] Diese Strahlen können Veränderungen im lebenden Organismus auslösen und Krebs verursachen. Insofern sollte eine dosisminimierte Durchführung der Aufnahmen garantiert werden.

Die Dauer einer Computertomographie hängt davon ab, welche Organe untersucht bzw. aufgenommen werden müssen. Der Patient muss allerdings erfahrungsgemäß mit einer Untersuchungs- und Aufnahmedauer von insgesamt 15 bis 30 Minuten rechnen, manchmal auch bis zu einer Stunde. Diese Zeit kann unter anderem bei der Beobachtung chemischer Reaktionen, die im Körper ablaufen, sehr entscheidend sein. Die Reaktionszeit kann sehr kurz sein, so dass die Aufnahme und die Darstellung sehr schnell erfolgen muss, um diese Reaktionen beobachten zu können.

Wie bereits erwähnt, werden die vom PET aufgefassten und unbearbeiteten Daten mit Hilfe einer Bildrekonstruktionsmethode in Querschnittbilder umgerechnet. Dieses Verfahren kann auch so verbessert werden, dass die Geschwindigkeit der Aufnahme und die Darstellung der Bilder sich erheblich verbessert. **Mit beachtlich rechenaufwändigeren iterativen Bildrekonstruktionsverfahren kann bei konstanter Bildqualität die für eine Untersuchung unumgängliche Strahlendosis dennoch um die Hälfte reduziert werden.**
Um das mangelhaft konditionierte Rekonstruktionsproblem zu lösen, gibt es zwei Klassen von Rekonstruktionsalgorithmen, auf die später genauer eingegangen wird:

- Die gefilterte Rückprojektion (FBP) und

4: http://www.welt.de/wissenschaft/article1667375/Experten_warnen_vor_Computertomografie.html

- die algebraische Rekonstruktionstechniken (ART).

1.2. Ziel der Arbeit

Im Mittelpunkt dieser Arbeit steht die Implementierung des Bresenham-Algorithmus in Matlab für die Benutzung in einem tomographischen Rekonstruktionsalgorithmus. Dieser wird für die Verringerung der nötigen Strahlendosis, eventuell auch für die Verbesserung der Geschwindigkeit und die damit verbundene Schwachstellen der Bildgebung verwendet. Durch diese Implementierung und mit Hilfe von den *„Baryzentrische Koordinaten“* können alle Voxel, die inmitten des Lichtbereiches einer bildgebende Methode liegen, ermittelt werden. Um im Anschluss die Bilder rekonstruieren zu können, wird der gesuchte relative bzw. exakte Volumenanteil für jedes Voxel berechnet.
Neben der exakten Volumenberechnung werden in dieser Arbeit ebenfalls erläutert wie das Voxelgewicht mit Hilfe der Länge, der ermittelten Voxel berechnet werden kann um Bilder mit etwas schlechterer Qualität schnell rekonstruieren zu können.

Ziel dieser Arbeit ist es ebenfalls, den Lesern und Leserinnen das Konzept hinter dem Bresenham-Algorithmus näher zu bringen. Des Weiteren ist wichtig, dass der Punkteinschlusstest Mittels baryzentrischer Koordinaten verdeutlicht wird.

1.3. Aufbau der Arbeit

Um einen perfekten Einstieg in die Algorithmus-Programmierung geben zu können, muss zunächst auf Matlab eingegangen werden. Es wird ein Überblick über die Tools und die Funktionen gezeigt. Denn diese Matlab-Funktionen sind nicht nur als Lösung mathematischer Probleme interessant zu wissen.
Im ersten Teil dieser Ausarbeitung werden die Grundlagen gelegt. Dort werden zwei Klassen von Rekonstruktionsalgorithmen erklärt. Der Punkteinschlusstest und die baryzentrische Koordinaten werden im zweiten Teil erläutert.

Auf den theoretischen Erkenntnissen der ersten beiden Teile folgt im dritten Teil die Implementierung des Bresenham-Algorithmus in Matlab und die Verwendung der baryzentrischen Koordinaten. Darauf aufbauend wird als letztes der Einsatz, der von Bresenham ermittelten Voxelkoordinaten erläutert.

2. Grundlagen

2.1. Begriffsabgrenzung Matlab

„MATLAB[5] ist eine Hochsprache und eine interaktive Umgebung, mit der Sie rechenintensive Aufgaben schneller als mit herkömmlichen Programmiersprachen wie C, C++ und Fortran ausführen können.

- Übersicht und Hauptmerkmale
- Entwicklung von Algorithmen und Anwendungen
- Datenanalyse und -zugriff
- Visualisieren von Daten
- Ausführen numerischer Berechnungen
- Publizieren von Ergebnissen und Implementieren von Anwendungen“. [Mathworks: Matlab, http://www.mathworks.de/products/matlab/, 02.03.2011]

2.2. Werkzeuge der Computertomographie

2.2.2. Die gefilterte Rückprojektion

Der häufigste Algorithmus, der in der tomographischen Rekonstruktion von klinischen Daten verwendet wird, ist die gefilterte Rückprojektion.

Wie in Abbildung 2 zu sehen ist, wird bei der Drehung der Kamera um einen Patienten, eine Reihe von planaren Bildern, die Projektionen genannt werden, erstellt. An jedem Halt passieren durch den Kollimator nur Photonen, die sich senkrecht zur Kamera Front bewegen. Da viele dieser Photonen aus verschiedenen Tiefen des Patienten stammen, ist das Ergebnis

5 Der Name MATLAB steht für Matrix Labor.

eine Überlagerung von allen Tracern emittierenden Organe entlang des spezifischen Wegs, ähnlich wie bei einem Röntgenbild, eine Überlagerung aller anatomischen Strukturen von drei Dimensionen in zwei Dimensionen.
Nachdem alle Projektionen akkumuliert wurden, werden die Projektionen nach den Schichten unterteilt. Alle Projektionen, die zu einer Schicht gehören, werden dann in ein Bild bestellt, welches Sinogram genannt wird. Ein Sinogram ist das resultierende Bild nach einer Radontransformation (s. Abb. 3).
Das Ziel des Rekonstruktionsprozesses ist es, den Radiotracer aus den Projektionsdaten wiederzugewinnen.

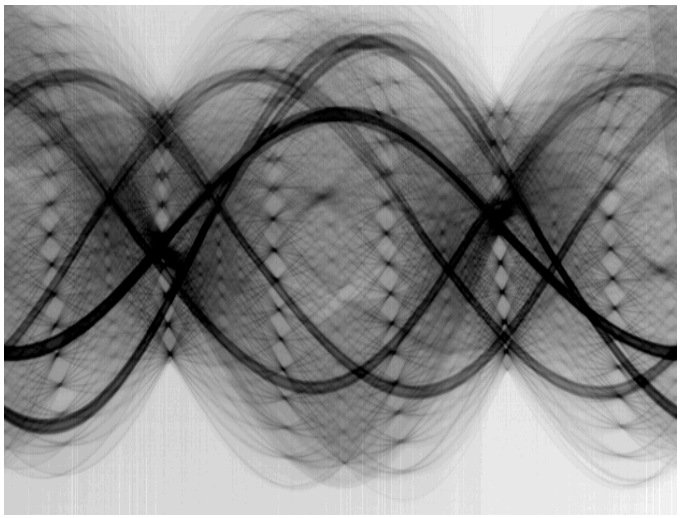


Abbildung 3: Sinogram

Quelle: http://www.ct.bam.de/images/0508sino.gif

Wenn an dieser Stelle die Sinogrammdaten rekonstruiert werden, würden aufgrund der Art der anschließenden Rückprojektion Artefakte erscheinen. Außerdem würde aufgrund der Radioaktivität Rauschen bei der Rekonstruktion entstehen. Anhand dieser beiden Effekte, ist es notwendig, die Daten zu filtern.
Um an dieser Stelle die Daten effizient zu filtern, werden die Projektionsdaten mit Hilfe der eindimensionalen Fourier-Transformation transformiert. Es stehen viele verschiedene Filter zur Verfügung, um die Daten zu filtern, und sie alle haben unterschiedliche Eigenschaften, auf die hier nicht eingegangen werden soll.
Diese Daten müssen im Anschluss, mit der eindimensionalen inversen Fourier-Transformation, rücktransformiert werden.

An dieser Stelle kann Rückprojektion durchgeführt werden. Wie am Anfang erwähnt, kommen durch den Kollimator nur Photonen, die sich senkrecht zur Kamera Front bewegen, durch. Die Rückprojektion schmiert die Daten, aus dem gefilterten Sinogram, wieder in die gleiche Richtung, von wo die Photonen ausgesendet wurden.[1][9]

Die gefilterte Rückprojektion ist somit „ein auf der Radon-Transformation beruhendes Verfahren zur Bildrekonstruktion, das in erster Linie in der Computertomographie verwendet wird." [*Quelle: Wikipedia, Gefilterte Rückprojektion,*[6]*, 20.02.2011*]

2.2.2. algebraische Rekonstruktionstechniken (ART)[7]

ART ist ein iterativer Algorithmus für die Rekonstruktion eines zweidimensionalen Bildes aus einer Serie von eindimensionalen Projektionen.
ART wurde von *Gordon*, *Bender*, und *Herman* als eine Methode für die Rekonstruktion von dreidimensionalen Objekten aus elektronenmikroskopischen Scans und Röntgenaufnahmen vorgeschlagen. Die Fourier-Methoden, die in dieser Zeit existierten, waren eher in der Rekonstruktion von Objekten begrenzt und hatten auch viele Anforderungen an die Rechnerleistung. Jedoch werden die Fourier-Methoden als einige der häufigsten Rekonstruktionsmethoden angesehen.

Eine der nennenswerten Vorteile gegenüber der Rückprojektion ist, dass ART bessere Resultate liefert als bei FBP wenn nicht genügend Projektionen für FBP vorliegen. Dies wiederum bedeutet eine geringe Bestrahlung des Patienten, was eines der Ziele gewesen ist.

In der algebraischen Methode ist die Rekonstruktion durch die Lösung eines lineares Gleichungssystems gegeben. Genauer gesagt kann ART als ein Problem der linearen Algebra geschrieben werden: $Wf = P$, wobei f ein unbekannter Spaltenvektor, wo die Dimension $d = (N \times 1)$ ist.

6 http://de.wikipedia.org/wiki/Gefilterte_R%C3%BCckprojektion

7 ART steht für **Algebraic Reconstruction Technique**

$f = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$, wo $N = n^2$ für Pixel in 2D oder $N = n^3$ für Voxel in 3D.

Das Bild ist als ein einzelner Punkt in einem N-dimensionalen Raum dargestellt. P ist die $(M \times 1)$ Spaltenvektor der p_i-Werte, das Linienintegral des i-ten Strahls repräsentieren (s. Abb. 4). M ist die Gesamtzahl der Strahlen in allen gemessenen Projektionen ist. Als Letztes ist W $(M \times N)$ Gewichts- bzw. Koeffizienten-Matrix, in der ein Element w_{ij}, also der Beitrag der j-ten Zelle im i-ten Strahl.
Die meisten w_{ij} sind Null, da nur eine geringe Anzahl von Zellen zu irgendeiner gegebenen Strahlsumme beitragen.

$$w_{ij} = \frac{vom\ Strahl\ i\ durchleuchtete\ Fläche\ von\ f_i}{Gesamtfläche\ von\ f_i}$$

Das lineare Gleichungssystem kann man wie folgt beschreiben:

$$\begin{aligned} w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + \cdots + w_{1N}f_N &= p_1 \\ w_{21}f_1 + w_{22}f_2 + w_{23}f_3 + \cdots + w_{2N}f_N &= p_2 \\ &\vdots \\ w_{M1}f_1 + w_{M2}f_2 + w_{M3}f_3 + \cdots + w_{MN}f_N &= p_M \end{aligned} \qquad \text{(Gl.1)}$$

In dem obigen linearen System stellt jede der oben genannten Gleichungen eine Hyperebene dar. Wenn nur eine Lösung für diese Gleichungen existiert, ist der Schnittpunkt aller Hyperebenen der Lösungspunkt. Und wenn M und N klein wären, könnten wir herkömmliche Matrixtheoriemethoden anwenden, um das System im oben genannten linearen System umzuwandeln. In der Praxis jedoch kann *N* 65.000 sein (für 256 × 256 Bilder), und i.d.R. haben M Strahlen in allen Projektionen auch die gleiche Größe für Bilder dieser Größe. In diesem Fall ist die Größe der Matrix $[w_{ij}]$ groß genug, um jede Möglichkeit der direkten Inversion entgegenzustehen. Bei Rauschen in den Messdaten und wenn $M < N$, auch für kleine N ist es dann nicht möglich eine direkte Matrixinversion zu

verwenden, in diesem Fall muss die „Methode der kleinsten Quadrate“ verwendet werden. Wenn $M > N$ gibt es in diesem Fall keine eindeutige Lösung.

Im allgemeinen Fall, wo M und N große Werte haben, existieren sehr attraktive iterative Methoden zur Lösung von (Gl.1), die Inversion ist da unpraktisch.
Um die Lösung des Gleichungssystems iterativ zu berechnen, kann beispielsweise die „Kaczmarz Methode“ verwendet werden. Jede Gleichung wird bei dieser Methode als n-1-dimensionale Hyperebene behandelt. In jedem Iterationsschritt wird ein Startvektor auf jeder dieser Hyperebenen projiziert.[5][4]

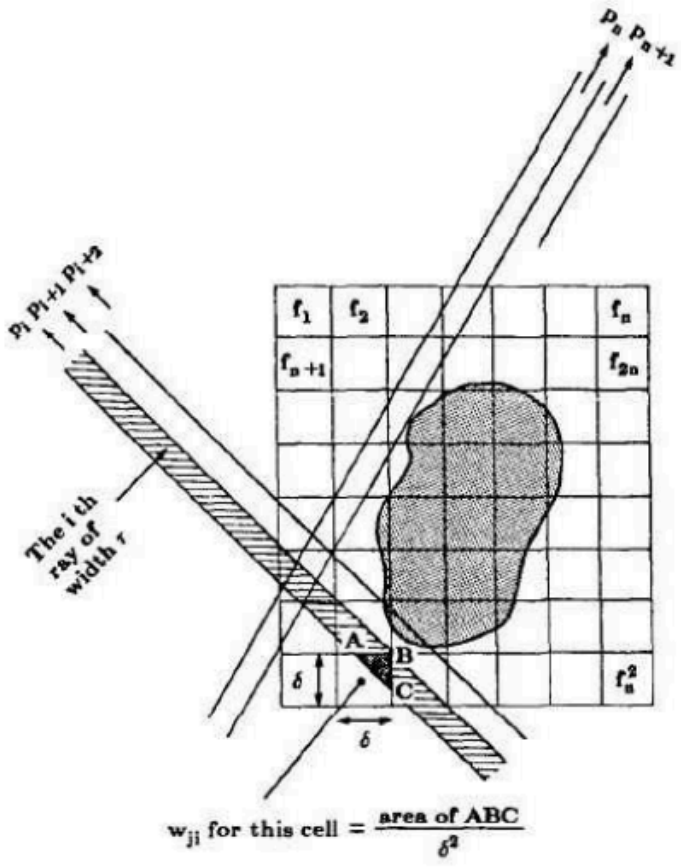


Abbildung 4: In ART a square grid is superimpsed over the unknown image. Image values are assumed to be constant within each cell of the grid.

Quelle: Digital Picture Processing, A.C.Kak, Malcoln Slaney

2.3. Baryzentrische Koordinaten

Der Begriff “Baryzentrisch“ wurde 1872 von *August Ferdinand Möbius* eingeführt. Die Baryzentrische Koordinaten dienen in der Geometrie und in der Linearen Algebra dazu die Eigenschaften eines Dreiecks zu untersuchen. Auch die Lage von Punkten bezüglich einem gegebenen Simplex, z.B. ein Dreieck oder eine Linie, können wir mit Hilfe dieser Koordinaten beschreiben.

Gegeben sei eine Menge von Punkten $P_0, P_1, P_2, ..., P_n$. Wenn für alle Punkte dieser Menge folgendes erfüllt ist,

$$\alpha_0 \boldsymbol{P_0} + \alpha_1 \boldsymbol{P_1} + \ldots + \alpha_n \boldsymbol{P_n}$$

wo

$$\alpha_0 + \alpha_0 + \ldots + \alpha_n = 1$$

so bildet diese Menge einen *affinen Raum,* und die Koeffizienten

$$(\alpha_0, \alpha_1, \ldots, \alpha_n)$$

werden die *baryzentrische Koordinaten* der Punkte im Raum genannt. [8]

Die baryzentrische Koordinaten fallen unter das Kapitel “Lineare Interpolation“ in der Linearen Algebra. Dort werden sie als „Längenverhältnisse auf einem Geradenabschnitt“ definiert [*8*]. Diese Koeffizienten bzw. Koordinaten sind sehr nützlich, vor allem bei dem Umgang mit Dreiecken.

Punkt auf einer Strecke

Um die baryzentrischen Koordinaten zu veranschaulichen, betrachten wir zwei Punkte P_1 und P_2 im Raum**.** Wenn die Koeffizienten α_1 und α_2 skalar sind und ihre Summe 1 beträgt, dann ist der Punkt *P* definiert durch einen Punkt auf der Strecke, die durch P_1 und P_2 läuft.

$$\boldsymbol{P = \alpha_1 P_1 + \alpha_2 P_2}$$

Der Punkt *P* liegt genau auf der Strecke zwischen P_1 und P_2 wenn gilt

$$0 \leq \alpha_1, \alpha_2 \geq 1.$$

Abbildung 5 zeigt ein Beispiel für eine Strecke und drei Punkte *P*, *Q*, *R*. Diese Punkte wurden mit folgenden Koeffizienten-Werten generiert**:**

$\boldsymbol{P}$**:** $\alpha_1 = \frac{1}{3}, \alpha_2 = \frac{2}{3}$

$\boldsymbol{Q}$**:** $\alpha_1 = \frac{3}{4}, \alpha_2 = \frac{1}{4}$

$\boldsymbol{R}$**:** $\alpha_1 = \frac{4}{3}, \alpha_2 = -\frac{1}{3}$

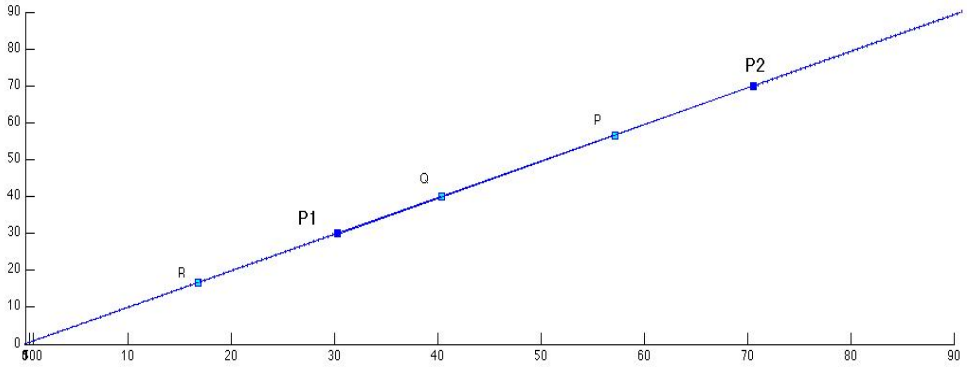


Abbildung 5

Punkt in einem Dreieck

Parallel dazu können wir die baryzentrischen Koordinaten für jedes beliebige Dreieck verwenden. Betrachtet wird ein beliebiger Punkt ***P***, der sich innerhalb von dem Dreieck ***T*** befindet. Der Punkt kann als Kombination der Ecken P_1, P_2 und P_3 wie folgt geschrieben werden:

$$\boldsymbol{P = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3}$$

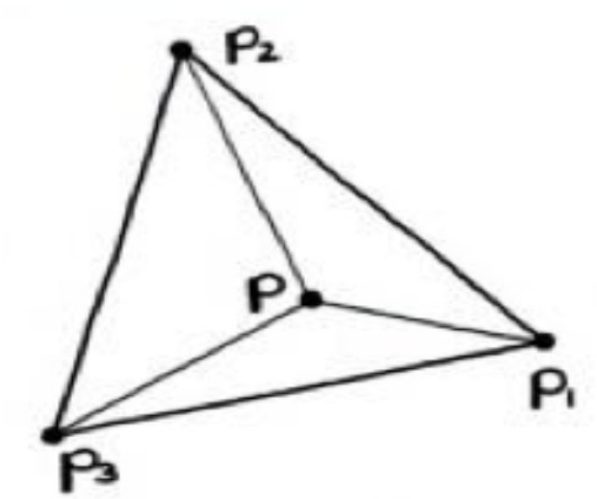


Abbildung 6: Baryzentrische Koordinaten

Quelle: „Lineare Algebra: Ein geometrischer Zugang“, Gerald Farin, D. Hansford, Seite 138

Die Summe der Koeffizienten der rechten Seite dieser Gleichung muss Eins ergeben, da es sich um eine Affinkombination handelt:

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \,.$$

Der Punkteinschlusstest ist die häufigste Anwendung bezüglich der baryzentrischen Koordinaten. Es handelt sich um die Frage ob sich Punkt ***P*** innerhalb des Dreiecks $\Delta\, \boldsymbol{P_1P_2P_3}$ befindet. Dies ist genau dann der Fall, wenn

$$0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1$$

gilt**.**

Wenn einer der drei Koeffizienten 0 ist, liegt *P* genau auf eine der drei Seiten des Dreiecks. Ansonsten befindet sich *P* außerhalb des Dreiecks.
Die Koeffizienten können ebenfalls durch die Flächenverhältnisse berechnet werden:

$$\alpha_1 = \frac{Flächeninhalt\ (P,P2,P3)}{Flächeninhalt\ (P1,P2,P3)},$$

$$\alpha_2 = \frac{Flächeninhalt\ (P,P3,P1)}{Flächeninhalt\ (P1,P2,P3)},$$

$$\alpha_3 = \frac{Flächeninhalt\ (P,P1,P2)}{Flächeninhalt\ (P1,P2,P3)}.$$

Diese Gleichungen bestätigen die Aussage, dass sich die drei Koeffizienten zu Eins addieren. Wir setzen $P = P_2$ und die Gleichungen sagen aus:

$\alpha_2 = 1$ und $\alpha_1 = \alpha_3$.

Eine weitere Bestätigung erfolgt, wenn *P* genau auf der Strecke [P_1 P_2] liegt. So ist das Ergebnis wie erwartet $\alpha_1 = 0$. Für die restlichen Ecken und Kanten gelten vergleichbare Beziehungen.

Damit können wir die baryzentrische Koordinaten der drei Ecken aufstellen:

$P_1 \cong (1,0,0)$,
$P_2 \cong (0,1,0)$,
$P_3 \cong (0,0,1)$.

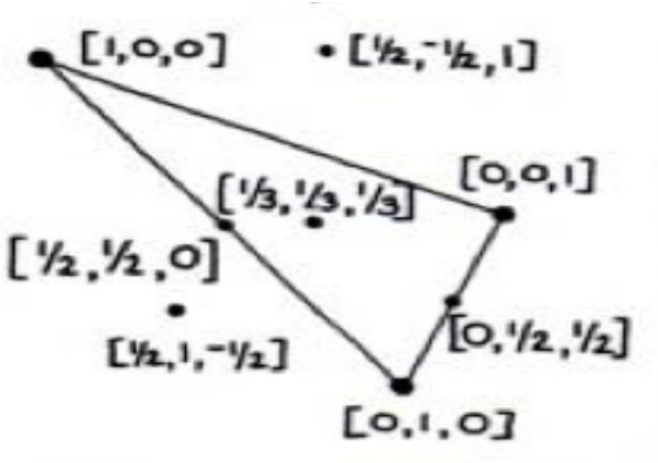


Abbildung 7: Verschiedene Beispiele baryzentrischer Koordinaten

Quelle: „Lineare Algebra: Ein geometrischer Zugang“, Gerald Farin, D. Hansford, Seite 139

MATLAB stellt für die baryzentrischen Koordinaten eine Funktion bereit, die ebenfalls auf Körper im Raum verwendet werden kann.

Die MATLAB-Funktion *tsearchn(X, TES, XI)* liefert die baryzentrischen Koordinaten von *XI*, wobei TES die *Delaunay-Triangulation* über alle Punkte in X ist. Die Funktion tsearchn liefert "NaN" wenn *XI* sich nicht innerhalb der konvexen Hüllen von X befindet. *XI* kann auch mehrere Punkte enthalten.

3. Bresenham-Algorithmus

Dieser Algorithmus ist der am besten geeignete, um Linien auf einem gerasterten Ausgabegerät, wie zum Beispiel einem Monitor, auszugeben. Zumindest was die Geschwindigkeit angeht, da keine Berechnungen mit Fließkommazahlen ausgeführt werden. Die genaue Position aller Pixel bzw. Voxel einer Linie können mit dem Bresenham'schen Linienalgorithmus bestimmt werden.

Der Bresenham-Algorithmus wird allgemein in den heutigen Grafikbeschleunigerchips verwendet. In dieser Arbeit wird, wie bereits erwähnt, dieser Algorithmus verwendet, um alle Voxeln, die durch einen Strahl getroffen werden, zu bestimmen.

Im Jahr 1962 entwickelte *Jack E. Bresenham* bei *IBM* seinen Algorithmus. 2001 schrieb er:

„I was working in the computation lab at IBM's San Jose development lab. A Calcomp plotter had been attached to an IBM 1401 via the 1407 typewriter console.
[The algorithm] was in production use by summer 1962, possibly a month or so earlier. Programs in those days were freely exchanged among corporations so Calcomp (Jim Newland and Calvin Hefte) had copies. When I returned to Stanford in Fall 1962, I put a copy in the Stanford comp center library.
A description of the line drawing routine was accepted for presentation at the 1963 ACM national convention in Denver, Colorado. It was a year in which no proceedings were published, only the agenda of speakers and topics in an issue of Communications of the ACM. A person from the IBM Systems Journal asked me after I made my presentation if they could publish the paper. I happily agreed, and they printed it in 1965.“ [*Quelle: Wikipedia, Bresenham´s line algorithm,*[8]*, 01.02.2011*]

8 http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

Der Bresenham-Algorithmus wurde später verändert um Kreise auszugeben. Der veränderte Algorithmus wurde als "Bresenham's circle algorithm" bekannt.

Arbeitsweise des *Algorithmus*

Um zu verstehen wie dieser Algorithmus arbeitet, , wird mit der allgemeinen Definition einer Linie, in der die Steigung das Verhältnis der *Veränderung* in Richtung der y-Achse zur *Veränderung* in Richtung der x-Achse darstellt, begonnen. Das wäre:

$$y = mx + b = \frac{\Delta y}{\Delta x}x + b. \qquad \text{(Gl.2)}$$

Die Konstante b ist der Wert von y, bei der die Linie die Y-Achse schneidet. Somit ist $(0, b)$ ein Punkt auf der Linie. Δx und Δy können, unter Verwendung von zwei verschiedenen Punkten auf der Linie berechnet werden. Wir können feststellen, dass die (Gl.2) y als Funktion von x darstellt, und dass sie jede Linie in der Fläche ausdrücken kann, außer einer vertikalen Linie, die im Wesentlichen einer endlosen Steigung entspricht. Vertikale Linien sind als Sonderfall zu behandeln. Alternativ können wir die Gleichung so ordnen, dass sie eine implizite Darstellung[9] der Linie liefert. Daraufhin ergibt sich:

$$F(x, y) = \Delta y x - \Delta x y + \Delta x b = 0. \qquad \text{(Gl.3)}$$

Diese Funktion spezifiziert die gleiche Linie, ist jedoch implizit. Das heißt: jeder Punkt (x, y), der die Bedingung *F(x, y) = 0* erfüllt, liegt auf der Linie. Allerdings legt *F* nicht fest, wie *y* bzw. x berechnet werden, wenn *x* bzw. y gegeben ist. Zusätzlich zur Spezifizierung der Linie übergibt *F* ein anderes Vorzeichen an den Punkten, die auf die andere Seite der Linie bzw. Gerade liegen. Und zwar $F(x,) < 0$, wenn (x, y) über die Linie liegt, und $F(x, y) > 0$, wenn (x, y) unter der Linie liegt (s. Abb. 8.a).

9 Siehe Otto Forster: *Analysis 2.* 7. Auflage. 2006, I.8 Implizite Funktionen, S. 86 - 99.

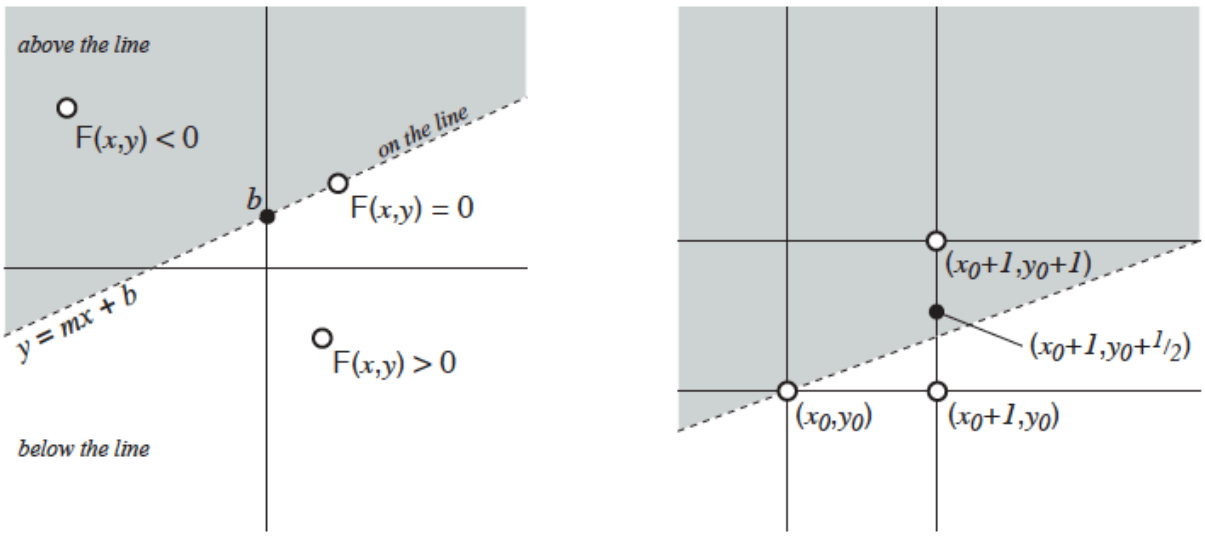


Abbildung 8: Eine Bildliche Darstellung der Bresenham-Algorithmus

Abbildung 8.a (links): Die Vorzeichenänderung von F(x,y)

Abbildung 8.b (rechts): (x0+1,y0+1/2) bildlich dargestellt

Quelle: http://www.scribd.com/doc/ICS183_Bresenhams_algorithm

Dieses ist die Eigenschaft, welche der Bresenham-Algorithmus ausnutzt, wenn er bestimmt, welcher Pixel am besten eine Linie approximiert.

Betrachtet wird nun eine Strecke, deren Steigung zwischen 0 und 1 liest, und von "links" *(x_0, y_0)* nach "rechts" *(x_1, y_1)* verläuft, sodass, $x_0 \leq x_1, y_0 \leq y_1$ und $y_1 - y_0 \leq x_1 - x_0$. Diese Einschränkung vereinfacht es, am Anfang den Algorithmus zu verstehen.

Sobald dieser Fall behandelt wurde, sind alle anderen Linien dementsprechend in einer ähnlichen Weise zu behandeln. Abbildung 8.b wird nun betrachtet. Diese zeigt den ersten Pixel auf der Linie bei *(x_0, y_0)*. Die erste "Entscheidung", dass die Algorithmus treffen muss ist: soll der nächste Punkt *(x_0 + 1, y_0)* oder *(x_0 + 1, y_0 + 1)* sein?

Diese sind die einzigen zwei logischen Möglichkeiten, da die Steigung der Gerade zwischen 0 und 1 ist. Um diese Entscheidung zu treffen, betrachten wir, ob der Punkt $(x_0 + 1, y_0 + \frac{1}{2})$, der sich in der Mitte zwischen den beiden Möglichkeiten befindet, „über“ oder „unterhalb“ der Linie liegt. Das heißt: wir betrachten das Vorzeichen von $f(x_0 + 1, y_0 + \frac{1}{2})$.

Der resultierende Wert von F wird die „Entscheidungsvariable“ oder „Fehlervariable“ genannt, da sie verwendet wird, um sich zwischen den zwei möglichen Pixeln zu entscheiden. Zu beachten ist, dass der Wert der Fehlervariable irrelevant ist, da nur das Vorzeichen erforderlich ist, um die Entscheidung zu treffen. Nach Einsetzen der Koordinaten

der zwischenliegenden Punkt ergibt sich:

$$\begin{aligned} F\left(x_0+1, y_0+\tfrac{1}{2}\right) &= \Delta y\,(x_0+1) - \Delta x\left(y_0+\tfrac{1}{2}\right) + \Delta x b \\ &= (\Delta y x_0 - \Delta x y_0 + \Delta x b) + (\Delta y - \tfrac{1}{2}\Delta x) \\ &= F(x_0, y_0) + (\Delta y - \tfrac{1}{2}\Delta x) \\ &= \Delta y - \tfrac{1}{2}\Delta x, \end{aligned} \qquad \text{(Gl.4)}$$

wobei $F(x_0, y_0) = 0$, im letzten Schritt verwendet wurde. Da der Wert selbst interessant ist, sondern nur das Vorzeichen, kann eine weitere Vereinfachung stattfinden, indem (Gl.3) mit 2 multipliziert wird. Somit ergibt sich die erste Fehlervariable

$$D = 2\Delta y - \Delta x. \qquad \text{(Gl.5)}$$

Die Fehlervariable in (Gl.5) ermöglicht die erste Entscheidung zu treffen. An dieser Stelle stellt sich die Frage, ob die folgenden Pixel den gleichen y-Wert wie die ersten haben oder dieser um 1 höher liegt. In den folgenden Schritten wird F erneut ausgerechnet und das Vorzeichen wird angeschaut. Es gibt jedoch eine etwas effizientere Methode, nämlich die „inkrementelle Berechnung“ bzw. die „Zuwachsberechnung“. Anstatt den Wert von F neu zu rechnen, wird berechnet inwiefern sich F von der vorhergehenden Fehlervariable unterscheidet.

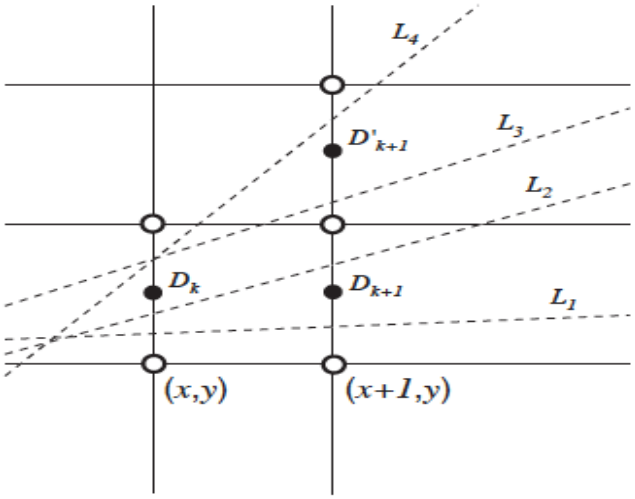


Abbildung 9

Quelle: http://www.scribd.com/doc/ICS183_Bresenhams_algorithm

Abbildung 9 stellt drei mögliche Entscheidungspunkte und ihre entsprechenden Fehlervariablen, D_k, D_{k+1} und D'_{k+1} bildlich dar. Wenn D_k schon berechnet wurde, unterscheidet sie sich von D_{k+1} und D'_{k+1} wie folgt:

$$F\left(x+2, y+\frac{3}{2}\right) - F\left(x+1, y+\frac{1}{2}\right) = \Delta y - \Delta x$$
$$F\left(x+2, y+\frac{3}{2}\right) - F\left(x+1, y+\frac{1}{2}\right) = \Delta y.$$

Um die Stufensprünge in Einklang mit der vorhergehenden Definition von D zu halten, müssen diese erneut mit 2 multiplizieret werden.[6][7]

Bresenham 3D und die Implementierung in Matlab

Für den 3D Linien-Algorithmus muss für die "neue" Z-Achse nur eine zweite Fehlervariable F_2 hinzugefügt werden. Kurz ausgedrückt:

$$F_2\left(x_0+1, z_0+\frac{1}{2}\right) = \Delta z - \frac{1}{2}\Delta x,$$
$$D_2 = 2\Delta z - \Delta x.$$

Um zu verstehen wie dieser Algorithmus in 3D arbeitet, wird zunächst betrachtet, wie ein Monitor einzelne Bildpunkte darstellt. Auf der Abbildung 10 ist gut ersichtlich, dass die "Bildfläche" in einzelne Quadrate aufgeteilt ist. Mit denen wird nun versucht eine Linie von A nach B zu zeichnen.

Um eine Linie zu zeichnen, werden zunächst zwei Punkte benötigt, nämlich der Anfangspunkt und der Endpunkt.

Matlab:

```
function [X,Y,Z] = bresenham (Pstart, Pziel)
```

Den Anfang der Linie wird hier auf diesen Punkt gesetzt

$$P_1 = \begin{pmatrix} x_{\text{start}} \\ y_{\text{start}} \\ z_{\text{start}} \end{pmatrix}$$

und das Ende auf

$$P_2 = \begin{pmatrix} x_{\text{ziel}} \\ y_{\text{ziel}} \\ z_{\text{ziel}} \end{pmatrix}.$$

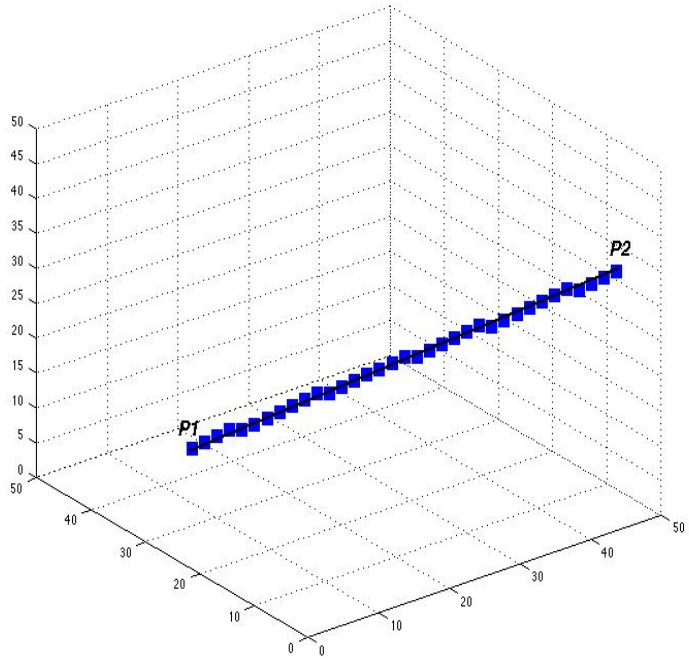


Abbildung 10

Außerdem werden für die Berechnung der einzelnen Punkte die drei Deltas gebraucht, die den Längen in den einzelnen Richtungen entsprechen:

$\Delta x = x_{\text{ziel}} - x_{\text{start}}$	deltaX = xziel - xstart;
$\Delta y = y_{\text{ziel}} - y_{\text{start}}$	deltaY = yziel - ystart;
$\Delta z = z_{\text{ziel}} - z_{\text{start}}$	deltaZ = zziel - zstart;

Um nun alle Linienarten auf dem Raster darstellen zu können, auch diejenigen Linien, die nicht von „links“ nach „rechts“ verlaufen bzw. deren Steigung nicht zwischen 0 und 1 ist, wird Schritt für Schritt vom Anfangspunkt der schnellen Richtung gegangen bis die

entsprechende Position des Endpunktes erreicht wird.

Die Richtung wird so ermittelt, indem der größte Wert der drei Deltas ermittelt wird:

$\Delta fast =$ max(deltaX, deltaY, deltaZ).

Die Richtung wird im folgenden als "*fast*" bezeichnet.

Für die einzelnen Schritte benötigt der Algorithmus die aktuelle X Position, die aktuelle Y Position, die aktuelle Z Position und zwei Fehlerwerte bzw. Fehlervariablen jeweils einen bzw. eine für die Richtungen, mit den langsamen Steigung. Diese Fehlerwerte müssen festgehalten und für jeden Schritt neu berechnet werden, um zu wissen wann ein Schritt in die andere "langsamen" Richtungen gehen muss.

Der Fehlerwert wird mit der Hälfte von $\Delta fast$ initialisiert:

```
if(deltaX >= max(deltaY,deltaZ))
yFehlerwert = deltaY - deltaX/2;
zFehlerwert = deltaZ - deltaX/2;
        ⋮
⋮
(Parallel für die anderen Richtungen)
```

Die aktuelle *fast* Position ist der vorherige Wert $+1$. Die anderen Positionen werden um 1 erhöht, wenn der Fehlerwert ≥ 0 wird. Der Fehlerwert ist immer der vorherige Wert abzüglich dem eben erreichten $\Delta fast$:

```
Falls X die schnelle Reichtung ist:
if(x == xziel)
  break;
end

if(yFehlerwert >= 0)
  y = y + 1;
  yFehlerwert = yFehlerwert - abstandX;
```

```
  end

  if(zFehlerwert >= 0)
    z = z + 1;
    zFehlerwert = zFehlerwert - abstandX;
  end

  x  = x  + 1;
  yFehlerwert = yFehlerwert + abstandY;
  zFehlerwert = zFehlerwert + abstandZ;
end
      ⋮
      (Parallel für die anderen Richtungen)
```

Ab hier wird immer der gleiche Prozess in jedem Schritt umgesetzt. Zunächst wird der aktuelle $fast$ Wert um 1 erhöht, dann werden die neuen Fehlerwerte berechnet. Wenn ein Fehlerwert ≥ 0 ist, erhöht sich die jeweilige Position um 1.
Der neue Voxel wird dann an der Stelle $X_{aktuell}$, $Y_{aktuell}$ und $Z_{aktuell}$ gezeichnet. So geht es Schritt für Schritt immer weiter. Dies geht solange bis P-aktuell P-ziel entspricht, nämlich genau dann ist die Linie fertig gezeichnet (vgl. Abb. 10).

4. Voxelgewicht Berechnung

Mit der Implementierung des Bresenham-Algorithmus können alle Voxel, die einen Strahl durchqueren, bestimmt werden. 3D ART kann auf Plausibilität überprüft werden, indem das Voxelgewicht, der ermittelten Voxel, berechnet wird.

Es gibt vier Begrenzungsgeraden für ein Pixel g_1, g_2, g_3 und g_4 . Die Quelle *Q* liegt im Ursprung, und die Mattscheibe befindet sich in Abstand *d* von der Quelle. Es gilt:

$$g_1: \vec{x} = \alpha_1 \begin{pmatrix} x' \\ d \\ z' \end{pmatrix},$$

$$g_2: \vec{x} = \alpha_2 \begin{pmatrix} x' + 1 \\ d \\ z' \end{pmatrix},$$

$$g_3: \vec{x} = \alpha_3 \begin{pmatrix} x' + 1 \\ d \\ z' + 1 \end{pmatrix},$$

$$g_4: \vec{x} = \alpha_4 \begin{pmatrix} x' \\ d \\ z' + 1 \end{pmatrix}.$$

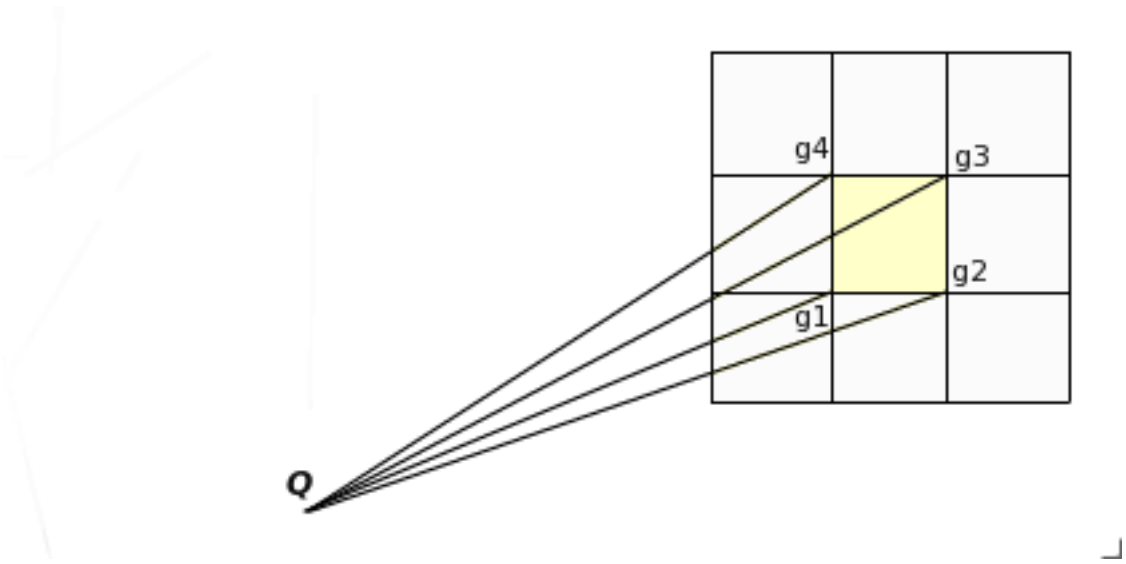


Abbildung 11: Die 4 Begrenzungsgeraden.

Die Begrenzungsgeraden sind Geraden an den vier Eckpunkten des Pixels, aufgrund der Quellenposition ist der Ortsvektor der Nullvektor. $x´, z´$ sind bekannt, da sie vom Bresenham-Algorithmus vorgegeben werden.

$g_0 = \alpha_0 \begin{pmatrix} x´ + \frac{1}{2} \\ d \\ z´ + \frac{1}{2} \end{pmatrix}$ ist eine Gerade im Mittelpunkt des Pixels. Wenn der Weg von g_0 im Voxel bekannt ist, kann das Voxelgewicht zu $w = \frac{\partial}{\sqrt{3}\varrho}$ berechnet werden, wenn ∂ den von g_0 im Voxel zurückgelegten Weg darstellt (v. Abb. 12) und ϱ die Kantenlänge des Voxels.

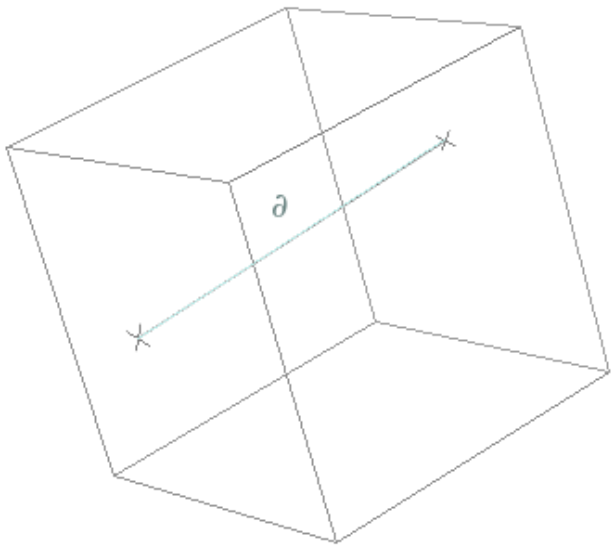


Abbildung 12: Darstellung von ∂, den von g_0 im Voxel zurückgelegen Weg

Um ∂ zu berechnen, müssen der Eintrittspunkt und der Austrittspunkt des Strahls im Voxel berechnet werden. Durch Gleichsetzen von Strahl- bzw. Geradengleichung und Ebenen-Gleichung entsteht ein LGS. Das Gauß-Jordan-Verfahren löst dieses LGS und liefert die benötigten Schnittpunkte.

5. Ergebnisse und Diskussion

Während der Erstellung dieser Arbeit wurde eine ausgedehnte und umfangreiche Literaturrecherche durchgeführt. Bei der Recherche wurden viele funktionierende und beschriebene Rasterungs-Verfahren gefunden. Weitere Verfahren im 3D Raum wurden nicht gefunden.

Der Bresenham-Algorithmus wurde im Rahmen dieser Arbeit weder für die Rasterung noch für die Zeichnung verwendet. Die Implementierung wurde für die Voxelermittlung verwendet. Demzufolge wurden die größten Nachteile von diesem Algorithmus vermieden, und zwar das Entstehen sogenannter „Treppenstufen“ oder Treppen-Effekte bei der Zeichnung. Vor allem bei Linien mit der Steigung 45 Grad, werden diese Linien unscharf.

Dagegen ist die Laufzeit von diesem Algorithmus linear, in anderen Worten: Die Laufzeit ist proportional zu der Länge des Strahls. Dabei wird kaum bzw. so wenig wie möglich Prozessorleistung erfordert.

Im Hinblick auf die bei der Tomographie eingesetzten Strahlen ist die Ausführungsdauer der Bresenham-Algorithmus für die Rekonstruktionsverfahren von Vorteil. Die Implementierung des Algorithmus kann dazu benutzt werden die Rekonstruktion gravierend zu beschleunigen. In einer Testumgebung bzw. Maschine (2,4 GHz Intel 2 core Prozessor, 4GB Arbeitsspeicher und MATLAB R2010b), liefen alle Testfälle bis auf eine Ausnahme problemlos und schnell durch (s. Testszenario). Die Darstellung der Strecke, zwischen der Quelle (0, 0, 0) und (100, 100, 100) lief fast genau so schnell wie die Darstellung der Strecke (0, 0, 0) – (10 000, 10 000, 10 000) mittels dem Bresenham-Algorithmus.

Die vorgestellte Implementierung des Algorithmus nach Bresenham kann so verbessert werden, indem die Implementation auch auf Gleitkommaeingaben ausführbar implementiert wird. So kann der Algorithmus deutlich genauer und ohne Treppen-Effekt

arbeiten. Dagegen spricht allerdings, dass die Gleitkommaarithmetik Zeit und Ressourcen verbraucht.

Ein Teilproblem der Algebraischen Rekonstruktionstechnik, das innerhalb dieser Arbeit von Interesse war, besteht darin, passende Gewichtungsfaktoren $0 \leq w_{ij} \leq 1$ zu finden, welche bei der Bestimmung der Gesamtemission miteinbezogen werden müssen (vgl. 2.2.2. Abb. 4). Für genaue und qualitätsreiche Bildrekonstruktion, ist die Berechnung des relativen Volumenanteils eines bestrahlten Voxels die erste Lösung. Die Bildrekonstruktion kann hingegen durch die Berechnung des zurückgelegten Wegs im Voxel schneller laufen (s. 4. Voxel Berechnung), wenn eine allerdings etwas niedrige Bildqualität zufriedenstellend ist. Mit dieser Voxelgewicht Berechnung kann 3D ART auf Plausibilität überprüft werden.

Im Rahmen dieser Arbeit werden die baryzentrischen Koordinaten für den Punkteinschlusstest benutzt. Den Zusammenhang zwischen den Bresenham-Algorithmus und die baryzentrischen Koordinaten, wird mit einem Beispiel besser veranschaulicht. Dieser Beispiel soll gleichzeitig als ein Testfall für die Implementierung des Bresenham-Algorithmus dienen.

Die Strahlquelle liegt im Ursprung, um einen Lichtbereich zu simulieren werden 4 Punkte im Raum gewählt und als Input für den implementierten Bresenham-Algorithmus übergeben.

```
P1=[0 0 0],             P2=[80 100 80],
P3=[100 100 80],        P4=[100 100 100] und
P5=[80 100 100].

[X1 Y1 Z1] = bresenham_line(P1, P2)
[X2 Y2 Z2] = bresenham_line(P1, P3)
```

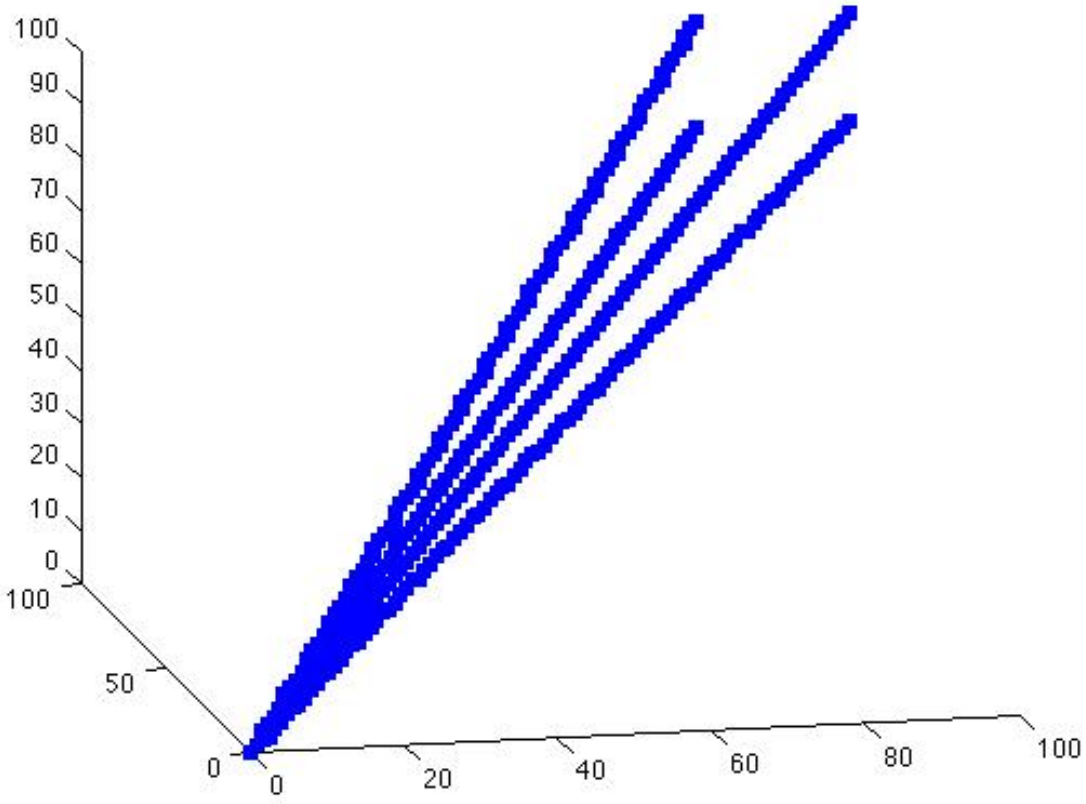
Damit gleichzeitig überprüft wird, ob der Algorithmus auch für links nach rechts verlaufende Lienien funktioniert, werden die Input-Punkte vertauscht.

```
[X3 Y3 Z3] = bresenham_line(P4, P1)
[X4 Y4 Z4] = bresenham_line(P5, P1)
```

Nachdem Aufruf werden alle ermittelten Voxel in [X Y Z] gespeichert. Die folgenden Befehle stellen die ermittelten Voxel in einem 3D-Koordinatensystem grafisch dar:

```
plot3(X1,Y1,Z1,'s','markerface','b');
hold on
plot3(X2,Y2,Z2,'s','markerface','b');
plot3(X3,Y3,Z3,'s','markerface','b');
```

```
plot3(X4,Y4,Z4,'s','markerface','b');
```



Nun werden vier Punkte zufällig definiert, um die baryzentrischen Koordinaten zu veranschaulichen:

```
testpunkt1 = [50 50 50],
testpunkt2 = [130 50 20],
testpunkt4 = [10 60 40],
testpunkt3 =[100 100 100].
```

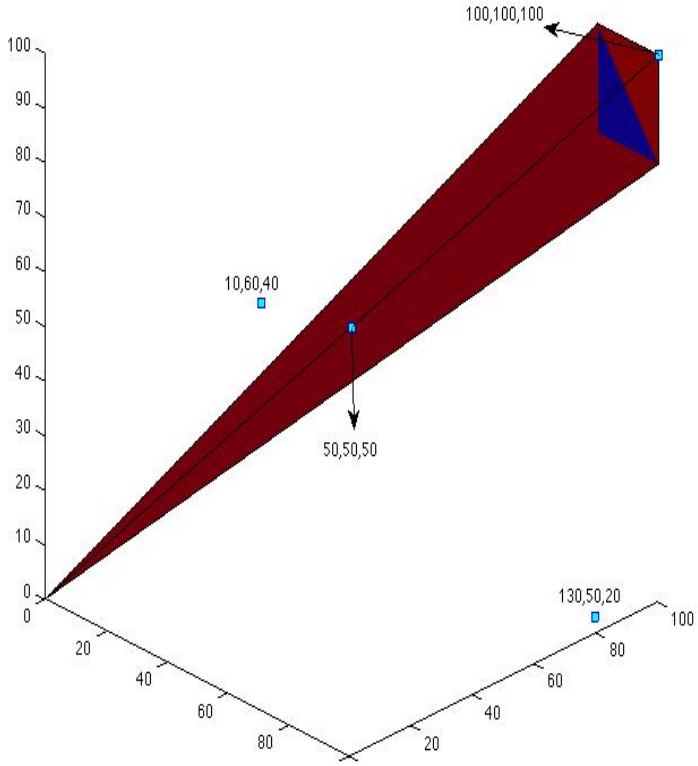
Danach muss mit Hilfe der Delaunay Triangulation über den Lichtbereich einige Dreiecke gebildet werden. Dafür wird die MATLAB-Funktion „DelaunayTri()“ aufgerufen, und als Input werden die ermittelten Voxel [X Y Z] übergeben. Es ist speicherschonend, wenn nur die Eckpunkte als Input übergeben werden, d.h. P1, P2, P3, P4 und P5:

```
TES = DelaunayTri([P1;P2;P3;P4;P5])
```

An diese Stelle können die baryzentrischen Koordinaten ermittelt werden:

```
tsearchn(X, TES, testpunkte), wobei X = [P1;P2;P3;P4;P5].
```

Für die Darstellung dient jetzt die Funktion `tetramesh()`:



Wie erwartet gibt tsearchn für die Testpunkte folgendes aus:

```
testpunkt1: ans = 2
testpunkt2: ans = NaN
testpunkt4: ans = NaN
testpunkt3: ans = 2.
```

Da „testpunkt2“ und „testpunkt4“ nicht im Strahlblickbereich liegen, werden sie außer Acht gelassen.

Testszenario*:*

Gegeben sei ein Objekt in Form von einem Würfel. Es müssen alle Voxel ermittelt werden, die sich innerhalb des Strahlbereiches befinden.

Der Würfel ist durch seine obere hintere rechte und untere vordere linke Ecken definiert:

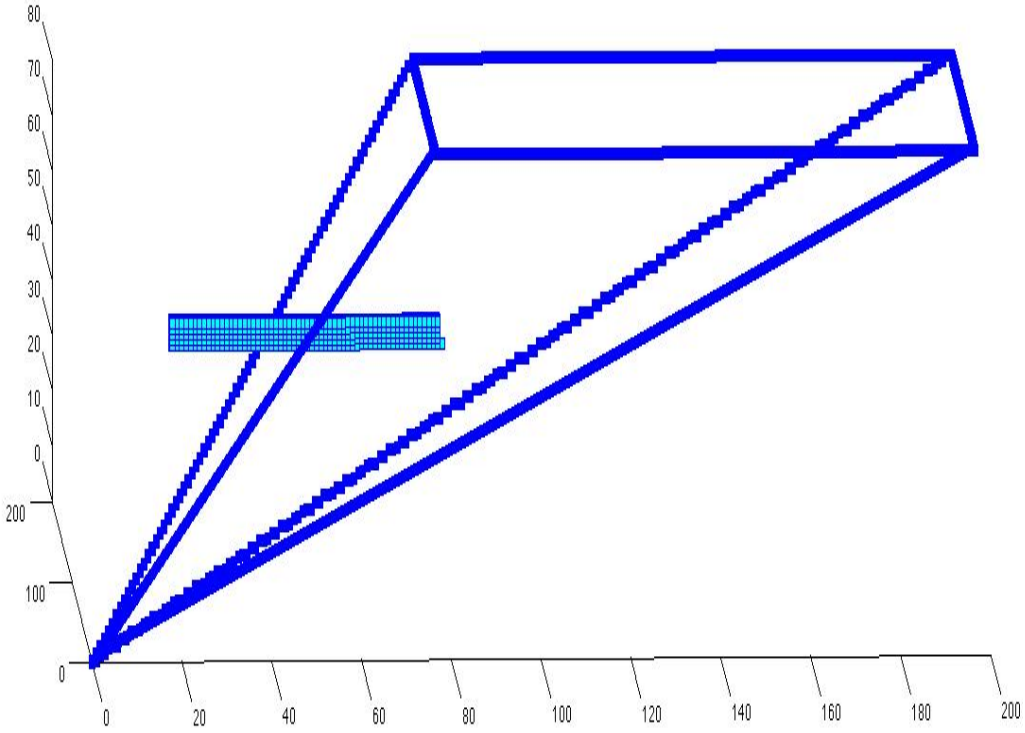
```
W1= [20 50 50];
W2= [45 75 75];.
```

Es müssen zuerst alle Würfelvoxel ermittelt werden. Aufgrund der Ausführzeit und die Computerressourcen werden, Parallel zur Ermittlung, die baryzentrischen Koordinaten aller Würfelvoxel berechnet und die Voxel grafisch dargestellt:

```
while W1(1) < P2(1)
    while W1(2) < W2(2)
        while W1(3) < W2(3)
             plot3(W1(1),W1(2),W1(3),'s','markerface','c');
             t=pointfinder([P1;P2;P3;P4;P5],W1);
             if t > 0;
                 treffer=[treffer;W1];
             end
             W1(3) = W1(3) + 1;
        end
        W1(3) = W12(3);
        W1(2) = W1(2) + 1;
        plot3(W1(1),W1(2),W1(3),'s','markerface','c');
    end
    W1(2) = W12(2);
    W1(1) = W1(1) + 1;
    plot3(W1(1),W1(2),W1(3),'s','markerface','c');
end .
```

Dieses Objekt kann auf die Testumgebung nicht dargestellt werden. MATLAB bleibt bei der gegebenen Würfel-Koordinaten hängen.

Durch Änderung der zweite Würfel-Koordinaten auf `W2= [25 55 55],` werden Würfelvoxel getroffen und durch tsearchn ausgegeben.


80
70
60
50
40
30
20
10
0
200
100
0
0
20
40
60
80
100
120
140
160
180
200

6. Literaturverzeichnis

[1] Erich Krestel*: Bildgebende Systeme für die medizinische Diagnostik*,1980

[2] *Heinz Handels: Medizinische Bildverarbeitung, Viewig+Teubner 2009*

[3] *Buzug Thorsten M: Einführung in die Computertomographie: Mathematisch-physikalische Grundlagen der Bildrekonstruktion,* Springer, 2002

[4] A. Rosenfeld and A. C. Kak: Digital Picture Processing, NY: Academic Press, 1982.

[5] *A. C. Kak and Malcolm Slaney, Principles of Computerized Tomographic Imaging, IEEE Press, 1988*

[6] Bresenham, J.E.: Algorithm for computer control of a digital plotter. In: *IBM Systems Journal* 4, 1965

[7] Wikipedia: *Bresenham-Algorithmus*. http://de.wikipedia.org/w/index.php?title=Bresenham-Algorithmus&oldid=70262651, 22.12.2010

[8] Gerald Farin, D. Hansford: Lineare Algebra: Ein geometrischer Zugang, Springer, S.136ff 2008

[9] *Torsten Rohlfing: Simulierte Computertomographie und vergleichende Bewertung verschiedener Rekonstruktionsverfahren, GRIN Verlag*

[10] Janser, A. ; Luther, W.: *Der Bresenham-Algorithmus*. Gerhard-Mercator-Universität-GH Duisburg, 1994

[11] Mathworks: http://www.mathworks.de/products/matlab/, 02.03.2011

7. Anhang

Anhang A: Bresenham-Algorithmus in Matlab

Anhang B: Baryzentrische Koordinaten in Matlab

Anhang C: Voxelgewicht Berechnung in Matlab

Anhang D: Schnittpunkt Berechnung in Matlab

Anhang A: Bresenham-Algorithmus in Matlab

```
% [X Y Z] = bresenham (P1, P2);
%
%  P1   - vector f¸r Punkt1, wo P1 = [x1 y1 z1]
%     (x1 y1 z1 k^nnen nur Integer sein)
%
%  P2   - vector for Point2, wo P2 =3D [x2 y2 z2]
%     (x2 y2 z2 k^nnen nur Integer sein)
%
%  X    - eine Reihe von x Koordinaten von Bresenham's linie
%  Y    - eine Reihe von y Koordinaten von Bresenham's linie
%  Z    - eine Reihe von z Koordinaten von Bresenham's linie
%
%  Alle voxel in XYZ set (d.h. P(i) =3D [X(i) Y(i) Z(i)]) wird die
%  Bresenham Linie darstellen
%
%  Beispiel:
%   P1 = [12 37 6];
%   P2 = [46 3 35];
%   [X Y Z] = bresenham_line3d(P1, P2);
%   figure; plot3(X,Y,Z,'s','markerface','b');
%
%  Beispiel2:
%   P1=[0 0 0];
%   P3=[100 100 80];
%   P2=[80 100 80];
%   P4=[100 100 100];
%   P5=[80 100 100];
%   [X1 Y1 Z1] = bresenham_line3d(P1, P2);
%   plot3(X1,Y1,Z1,'s','markerface','b');
%   hold on
%   [X2 Y2 Z2] = bresenham_line3d(P1, P3);
%   plot3(X2,Y2,Z2,'s','markerface','b');
%   [X3 Y3 Z3] = bresenham_line3d(P1, P4);
%   plot3(X3,Y3,Z3,'s','markerface','b');
%   [X4 Y4 Z4] = bresenham_line3d(P1, P5);
%   plot3(X4,Y4,Z4,'s','markerface','b');

function [X,Y,Z] = bresenham_line3d(Panfang, Pziel)

   xanfang = Panfang(1);
   yanfang = Panfang(2);
   zanfang = Panfang(3);

   xziel = Pziel(1);
   yziel = Pziel(2);
   zziel = Pziel(3);

   deltaX = xziel - xanfang;
   deltaY = yziel - yanfang;
   deltaZ = zziel - zanfang;

   abstandX = abs(deltaX)*2;
   abstandY = abs(deltaY)*2;
   abstandZ = abs(deltaZ)*2;
```

```
   sx = sign(deltaX);
   sy = sign(deltaY);
   sz = sign(deltaZ);

   x = xanfang;
   y = yanfang;
   z = zanfang;
   arrayIndex = 1;

   if(abstandX >= max(abstandY,abstandZ))           % x dominant, da dx am
gr^flten ist
      yFehlerwert = abstandY - abstandX/2;
      zFehlerwert = abstandZ - abstandX/2;

      while(1)
         X(arrayIndex) = x;
         Y(arrayIndex) = y;
         Z(arrayIndex) = z;
         arrayIndex = arrayIndex + 1;

         if(x == xziel)     % ende, ziel an der X-achse erreicht
            break;
         end

         if(yFehlerwert >= 0)       % entlang y bewegen
            y = y + sy;
            yFehlerwert = yFehlerwert - abstandX;
         end

         if(zFehlerwert >= 0)       % entlang z bewegen
            z = z + sz;
            zFehlerwert = zFehlerwert - abstandX;
         end

         x  = x  + sx;      % entlang x bewegen
         yFehlerwert = yFehlerwert + abstandY;
         zFehlerwert = zFehlerwert + abstandZ;
      end

   elseif(abstandY>=max(abstandX,abstandZ))     % y dominant, da dy am gr^flten
ist
      xFehlerwert = abstandX - abstandY/2;
      zFehlerwert = abstandZ - abstandY/2;

      while(1)
         X(arrayIndex) = x;
         Y(arrayIndex) = y;
         Z(arrayIndex) = z;
         arrayIndex = arrayIndex + 1;

         if(y == yziel)     % ende, ziel an der Y-achse erreicht
            break;
         end

         if(xFehlerwert >= 0)       % entlang x bewegen
            x = x + sx;
```

```
            xFehlerwert = xFehlerwert - abstandY;
         end

         if(zFehlerwert >= 0)       % entlang z bewegen
            z = z + sz;
            zFehlerwert = zFehlerwert - abstandY;
         end

         y  = y  + sy;      % entlang y bewegen
         xFehlerwert = xFehlerwert + abstandX;
         zFehlerwert = zFehlerwert + abstandZ;
      end

   elseif(abstandZ>=max(abstandX,abstandY))     % z dominant, da dz am grˆflten
ist
      xFehlerwert = abstandX - abstandZ/2;
      yFehlerwert = abstandY - abstandZ/2;

      while(1)
         X(arrayIndex) = x;
         Y(arrayIndex) = y;
         Z(arrayIndex) = z;
         arrayIndex = arrayIndex + 1;

         if(z == zziel)     % ende, ziel an der Z-achse erreicht
            break;
         end

         if(xFehlerwert >= 0)       % entlang x bewegen
            x = x + sx;
            xFehlerwert = xFehlerwert - abstandZ;
         end

         if(yFehlerwert >= 0)       % entlang y bewegen
            y = y + sy;
            yFehlerwert = yFehlerwert - abstandZ;
         end

         z  = z  + sz;      % entlang z bewegen
         xFehlerwert = xFehlerwert + abstandX;
         yFehlerwert = yFehlerwert + abstandY;
      end
   end

   return;                  % bre
```

Anhang B: Baryzentrische Koordinaten in Matlab

```
function [ t ] = pointfinder( X , p)
%POINTFINDER Diese Funktion ¸berpr¸ft ob p sich innerhalb der konvexe H¸lle
%von X befindet. Liefert NaN falls sich der punkt ausserhalb befindet

TES = DelaunayTri(X);

t=tsearchn(X, TES, p);

%tetramesh(TES);

end
```

Anhang C: Voxelgewicht Berechnung in Matlab

```
function [ vgewicht ] = voxelgewicht( eintrittspunkt, austrittspunkt,
kantenlaenge )
% diese funktion berechnet das Voelgewicht
% als Input bekommt diese Funktion die beiden schnittpunkte, die von der
% funktion "schnittpunkte" geliefert werden. und die Kantenl‰nge
% zur¸ck kommt das voxelgewicht

p=austrittspunkt - eintrittspunkt;
strahllaenge = sqrt(p(1)^2+p(2)^2+p(3)^2)
vgewicht = strahllaenge/(sqrt(3)*kantenlaenge);

end
```

Anhang D: Schnittpunkt Berechnung in Matlab

```
% Script-File: voxelebene
%
%Diese Funktion bekommt als input eine Voxelebene(Nr_Voxelebene) so wie
%die Richtungen innerhalb eines Voxels(RV) dieser Ebene und liefert dann die
beiden
%Spannvektoren und den Ortsvektor dieser Ebene zur¸ck.
%
%NR_Voxelebene--> 1=vorne, 2=rechts, 3=hinten, 4=links, 5=oben, 6=unten
%OV-->            Ursprung des Voxels
%RV-->            [RV(:,1), RV(:,2),RV(:,3)]
%v --> die Position des Voxels-3X1 zb[3;2;1]
%
%zum Beispiel:
%[Ortsvektor,Spannvektoren]=ortsvektor_spannvektoren(5,[100;100;100],[[1;0;0],
[0;1;0],[0;0;1]], [3;2;1])
%
%liefert folgendes zur¸ck;
%Ortsvektor =[;;]
%Spannvektoren =[[;;],[;;]]

function
[Ortsvektor,Spannvektoren]=ortsvektor_spannvektoren(Nr_Voxelebene,OV,RV,v)

if(Nr_Voxelebene==1) %vordere Voxelebene
        %Spannvektoren = [[1;0;0], [0;0;1]];
        Spannvektoren =[RV(:,1),RV(:,3)];

        Ecke = [0;0;0]; % links unten vorne
        Ortsvektor =OV+ RV*(v+Ecke);



elseif(Nr_Voxelebene==2) %Voxelebene rechts
        %Spannvektoren = [[0;-1;0], [0;0;-1]];
        Spannvektoren =(-1)*[RV(:,2),RV(:,3)];

        Ecke =[1;1;1]; % rechts oben hinten
        Ortsvektor =OV+ RV*(v+Ecke);



elseif(Nr_Voxelebene==3) %Voxelebene hinten
        %Spannvektoren = [[-1;0;0], [0;0;-1]];
        Spannvektoren =(-1)*[RV(:,1),RV(:,3)];

        Ecke =[1;1;1]; % rechts oben hinten
        Ortsvektor =OV+ RV*(v+Ecke);



elseif(Nr_Voxelebene==4) %Voxelebene links
        %Spannvektoren = [[0;1;0],[0;0;1]];
        Spannvektoren =[RV(:,2),RV(:,3)];
```

```
        Ecke =[0;0;0]; % links unten vorne
        Ortsvektor =OV+ RV*(v+Ecke);


elseif(Nr_Voxelebene==5) %Voxelebene oben
        %Spannvektoren = [[-1;0;0],[0;-1;0]];
        Spannvektoren =(-1)*[RV(:,1),RV(:,2)];

        Ecke =[1;1;1]; % rechts oben hinten
        Ortsvektor =OV+ RV*(v+Ecke);


else
        6; % Voxelebene unten
        %Spannvektoren = [[1;0;0],[0;1;0]];
        Spannvektoren =[RV(:,1),RV(:,2)];

        Ecke = [0;0;0]; % links unten vorne
        Ortsvektor =OV+ RV*(v+Ecke);

end

end
```

```
% Script-File: voxelebene
%
%Diese Funktion bekommt als input eine Voxelebene(Nr_Voxelebene) so wie
%die Richtungen innerhalb eines Voxels(RV) dieser Ebene und liefert dann die
beiden
%Spannvektoren und den Ortsvektor dieser Ebene zur¸ck.
%
%NR_Voxelebene--> 1=vorne, 2=rechts, 3=hinten, 4=links, 5=oben, 6=unten
%OV-->            Ursprung des Voxels
%RV-->            [RV(:,1), RV(:,2),RV(:,3)]
%v --> die Position des Voxels-3X1 zb[3;2;1]
%
%zum Beispiel:
%[Ortsvektor,Spannvektoren]=ortsvektor_spannvektoren(5,[100;100;100],[[1;0;0],
[0;1;0],[0;0;1]], [3;2;1])
%
%liefert folgendes zur¸ck;
%Ortsvektor =[;;]
%Spannvektoren =[[;;],[;;]]

function
[Ortsvektor,Spannvektoren]=ortsvektor_spannvektoren(Nr_Voxelebene,OV,RV,v)

if(Nr_Voxelebene==1) %vordere Voxelebene
        %Spannvektoren = [[1;0;0], [0;0;1]];
        Spannvektoren =[RV(:,1),RV(:,3)];
```

```
        Ecke = [0;0;0]; % links unten vorne
        Ortsvektor =OV+ RV*(v+Ecke);



elseif(Nr_Voxelebene==2) %Voxelebene rechts
        %Spannvektoren = [[0;-1;0], [0;0;-1]];
        Spannvektoren =(-1)*[RV(:,2),RV(:,3)];

        Ecke =[1;1;1]; % rechts oben hinten
        Ortsvektor =OV+ RV*(v+Ecke);



elseif(Nr_Voxelebene==3) %Voxelebene hinten
        %Spannvektoren = [[-1;0;0], [0;0;-1]];
        Spannvektoren =(-1)*[RV(:,1),RV(:,3)];

        Ecke =[1;1;1]; % rechts oben hinten
        Ortsvektor =OV+ RV*(v+Ecke);



elseif(Nr_Voxelebene==4) %Voxelebene links
        %Spannvektoren = [[0;1;0],[0;0;1]];
        Spannvektoren =[RV(:,2),RV(:,3)];

        Ecke =[0;0;0]; % links unten vorne
        Ortsvektor =OV+ RV*(v+Ecke);


elseif(Nr_Voxelebene==5) %Voxelebene oben
        %Spannvektoren = [[-1;0;0],[0;-1;0]];
        Spannvektoren =(-1)*[RV(:,1),RV(:,2)];

        Ecke =[1;1;1]; % rechts oben hinten
        Ortsvektor =OV+ RV*(v+Ecke);


else
        6; % Voxelebene unten
        %Spannvektoren = [[1;0;0],[0;1;0]];
        Spannvektoren =[RV(:,1),RV(:,2)];

        Ecke = [0;0;0]; % links unten vorne
        Ortsvektor =OV+ RV*(v+Ecke);

end

end



%-Script-File: Schnittpunkte
```

```
%
%Diese Funktion liefert alle Schnittpunkte der vier "Begrenzungsgeraden"
%(R^ntgenstrahlen) und der sechs Ebenen des kleinsten Elements eines
%K^rpers(Voxel)
%R_Strahl --> Richtungsvektor der Geraden, zb. [3;8;2]
%RV --> Richtungsvektoren der sechs Voxelebenen
%v ist die position des voxels im K^rper
%
%zum Beispiel;
%Eingabe:
%[Schnittpunkte]=schnittpunkte([100;100;100],[[1;0;0],[0;1;0],[0;0;1]],[80;80;
79])
%zur¸ck geliefert werden:
%alle Schnittpunkte in Form einer Matrix.

function [Schnittpunkte]  = schnittpunkte(R_Strahl,RV,v)

d=[];
Schnittpunkte=d;
%plot3(v(1),v(2),v(3),'s','markerface','c');
%hold on;

   for VoxelebeneNR=1:6%laufe alle sechs Voxelebenen durch

        %Bestimmung der Ebenengleichung:
        %Ebenengl.allg: Ortsvektor_E + r*Spannvektor_E(1) + s*Spannvektor_E(2)

[Ortsvektor_E,Spannvektor_E]=ortsvektor_spannvektoren(VoxelebeneNR,[0;0;0],RV,
v);

        %Geradengleichung allg: O + t*R_Strahl
        %Ortsvektor O f‰llt weg, da es der Nullvektor ist.
        %demnach lautet die G.gl. g = t*R_Strahl
        %durch Gleichsetzen von Gerade und Ebene entsteht folgendes LGS
        % t*R_Strahl = Ortsvektor_E +r*Spannvektor_E(1)+ s*Spannvektor_E(2)
        % Sortieren nach Unbekannten r, s und t;
        % r*Spannvektor_E(1)+ s*Spannvektor_E(2)-t*R_Strahl = Ortsvektor_E

  A=[Spannvektor_E(:,1),Spannvektor_E(:,2),-R_Strahl,-Ortsvektor_E];
  A=rref(A);%(rref steht fuer reduced row echelon form,reduzierte
  %Stufenform) fuehrt eine Variante des Eliminationsverfahrens durch (Gaufl-
  %Jordan Verfahren), allerdings nur mit Spalten-Pivotsuche.



  %Die Ergebnisse des LGS 'interpretieren' und Schnittpunkte liefern
  if(A(3,3)==0) %das LGS ist nicht eindeutig, da x=0


    %ueberpruefe nun, ob eine Nullzeile (t=0) vorliegt oder nicht
    if(A(3,4)==0) %beliebig viele L^sungen; quasi eine Nullzeile,
    else % keine Lsg bzw.Widerspruch, Gerade-Ebene "echt parallel" zueinander
    end


  else  %genau eine L^sung-->Schnittpunkt vorhanden
   %Bestimmung der Parameter
```

```
    r=A(1,4);
    s=A(2,4);
    t=A(3,4);

  end


  if(r>=0 && r<=1 && s<=1 && s>=0 && t<=1 && t>=0)
    %Einsetzen von t in die Geradengl., um den Schnittpunkt zu ermitteln
    %alle vorhandenen Schnittpunkte der Reihe nach in die Matrix schreiben
    d2=Ortsvektor_E + r*Spannvektor_E(1) + s*Spannvektor_E(2);


    d=[d, d2];

    %plot3([0;d2(1)],[0;d2(2)],[0;d2(3)]); grid on;

    Schnittpunkte=d;



  end
  end
end
```